uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería Mecánica
2018-2019

Trabajo Fin de Grado

"Estudio de dinámica por ordenador - MUBODYNA"

Jaime San Martín Puy

Tutor: Eduardo Corral Abad

[Incluir en el caso del interés de su publicación en el archivo abierto]

Trabajo de Fin de Grado
Grado en Ingeniería Mecánica

Título: ESTUDIO DE DINÁMICA POR ORDENADOR – MUBODYNA.

Autor: Jaime San Martin Puy.

Tutor: Eduardo Corral Abad.

TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día _____de____ _____ de 20_____ en Leganés, en la Escuela Politécnica Superior de Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

SECRETARIO VOCAL PRESIDENTE

RESUMEN

El presente trabajo de Fin de Grado de Ingeniería Mecánica tiene como objetivo el estudio y análisis de dinámica de mecanismos por ordenador. Para alcanzar dicho objetivo se comienza por el análisis y estudio de los métodos que se conocen hoy en día para el modelado y desarrollo de mecanismos, para su posterior aplicación en un MUBODYNA. MUBODYNA es un conjunto de algoritmos desarrollados e implementados en Matlab por la Universidad de Mihno en Portugal.

Tras la explicación de los diferentes métodos se desarrollarán varios modelos en el programa, con el fin de complementar el manual ya existente de MUBODYNA, y así facilitar el desarrollo del programa en investigaciones futuras.

Este trabajo se ha desarrollado con la idea de expandir y complementar el trabajo "*Dinámica por ordenador con Matlab*" [1], con el fin de facilitar el trabajo de investigación, tanto para alumnos como cualquier investigador, al disponer de varios modelos desarrollados con el programa, además de explicaciones sobre cómo definir los diferentes sistemas mecánicos que se pueden encontrar, y el proceso de cálculo que lleva a cabo la simulación.

Palabras clave:

Dinámica de mecanismos; Dinámica por ordenador; Análisis, Simulación.

ABSTRACT

The present Mechanical Engineering project has as objective the study and analysis of dynamics of mechanisms by computer. To reach this objective, we begin by analyzing and studying the methods that are known today for the modeling and development of mechanisms, for their subsequent application in MUBODYNA. MUBODYNA is a set of algorithms developed and implemented in Matlab by the University of Mihno in Portugal.

After the explanation of the different methods, several models will be developed in the program, in order to complement the existing manual of MUBODYNA, and thus facilitate the development of the program in future research.

This project has been developed with the idea of expanding and complementing the work "*Dinámica por ordenador con Matlab*" [1], in order to facilitate the research work, both for students and any researcher, by having several models developed with the program, in addition to explanations on how to define the different mechanical systems that can be found, and the calculation process carried out by the simulation.

Keywords:

Dynamics of mechanisms; Computer dynamics; Analysis, Simulation.

Índice

CAPÍTULO 0. MOTIVACIÓN Y OBJETIVOS 1

CAPÍTULO I: MODELADO SISTEMAS MECÁNICOS 2

1. Introducción 2

1.1 Sistema Mecánico 2

2. Conceptos Fundamentales 3

2.1 Grados de libertad 3

2.1.1 Caso plano 4

2.1.2 Caso tridimensional 4

2.2 Coordenadas 6

2.2.1 Coordenadas independientes 6

2.2.2 Coordenadas dependientes 7

2.2.2.1 Coordenadas relativas 7

2.2.2.2 Coordenadas de punto de referencia 8

2.2.2.3 Coordenadas naturales 10

3. Definición del sistema 11

3.1 Coordenadas globales y locales 11

3.2 Ángulos de Euler y parámetros de Euler 13

3.3 Juntas cinemáticas 18

3.3.1 Junta esférica 18

3.3.2 Junta de revolución 19

CAPÍTULO II: MODELOS DESARROLLADOS EN MUBODYNA 20

4. Péndulo - Esfera - Pared 20

5. Esfera-esfera 28

6. Deslizadera 34

7. Comandos save y load 43

CONCLUSIONES 45

PRESUPUESTO DEL PROYECTO 46

BIBLIOGRAFÍA 47

Índice de figuras

Figura 1. Multiboy System o Sistema Mecánico 3

Figura 2. Dirección de un automóvil 3

Figura 3. Mecanismo RSCR 4

Figura 4. Mecanismo 3 barras 5

Figura 5. Tipos de coordenadas 6

Figura 6. Coordenadas independientes 7

Figura 7. Coordenadas relativas 8

Figura 8. Coordenadas de punto de referencia 9

Figura 9. Coordenadas naturales 10

Figura 10. Definición de un punto 12

Figura 11. (a) Traslación y rotación (b) Solo rotación 13

Figura 12. Estados de rotación 15

Figura 13. Eje de rotación 16

Figura 14. Casos especiales en la rotación que simplifican los parámetros de Euler 17

Figura 15. Junta esférica 18

Figura 16. Junta de revolución 19

Figura 17. Definición cuerpos 20

Figura 18. Definición de cuerpos 21

Figura 19. Definición de puntos 22

Figura 20. Definición de puntos 22

Figura 21. Definición de juntas 23

Figura 22. Definición de fuerzas 24

Figura 23. Definición de fuerzas 24

Figura 24. inAnimate.m 25

Figura 25. inSolver.m 26

Figura 26. Simulación 27

Figura 27. Simulación 27

Figura 28. Definición cuerpos 28

Figura 29. Definición cuerpos 29

Figura 30. Definición de puntos 29

Figura 31. Definición de puntos 30

Figura 32. Definición de vectores de tipo 2 30

Figura 33. Definición de juntas 30

Figura 34. Definición de fuerzas 31

Figura 35. Definición de fuerzas 32

Figura 36. Definición de fuerzas 32

Figura 37. inAnimate 33

Figura 38. inSolver 33

Figura 39. Simulación 34

Figura 40. inBodies 35

Figura 41. inBodies 35

Figura 42. inPoints 36

Figura 43.inPoints 37

Figura 44. inVectors1 37

Figura 45. inVectors2 38

Figura 46. inFuncts 38

Figura 47.inJoints 39

Figura 48.inJoints 40

Figura 49. inForces 40

Figura 50. inAnimate 41

Figura 51. inSolver 41

Figura 53. Simulación 42

Figura 52. Simulación 42

Figura 54. Add to path 44

Figura 55. Comandos save and load 44

CAPÍTULO 0. MOTIVACIÓN Y OBJETIVOS

El objetivo del proyecto es el estudio y simulación de sistemas mecánicos mediante el programa MUBODYNA. Este es un conjunto de algoritmos implementados en Matlab, desarrollados en la Universidad de Mihno (Portugal), que permiten la simulación y análisis de conjuntos mecánicos. La Universidad Carlos III de Madrid colabora en el desarrollo de estos algoritmos, mediante investigaciones y estudios que ayuden en la mejora de la simulación.

La gran ventaja de estos programas de simulación es que son capaces de predecir cómo se va a comportar un sistema mecánico. MUBODYNA tiene un gran ámbito de aplicación, capaz de simular, por ejemplo, mecanismos o sistemas que impliquen choques e impactos. Se pueden implementar mecanismos que se emplean en la vida cotidiana, como los sistemas de dirección de los coches, sistema de suspensión, o modelos más sencillos como el funcionamiento de una puerta.

En una fase de diseño de un producto o sistema, la simulación del comportamiento es de gran importancia para comprobar, de manera aproximada, como se va a comportar el sistema. Durante el proyecto se explicarán conceptos utilizados por estos programas para llevar a cabo la simulación.

Los objetivos de este proyecto se pueden dividir en dos partes:

- Una primera parte de entendimiento de los procesos para definir un sistema mecánico, en el que se tratará de adquirir los conocimientos del necesarios para llevar a cabo la definición y simulación de estos sistemas.
- La segunda parte, y más importante, se trata de estudiar el funcionamiento de MUBODYNA, de todos los archivos que contiene, y ser capaces de desarrollar la capacidad para definir modelos en el programa correctamente.

A su vez, se explicarán conceptos que puedan servir de ayuda y orientación para futuras investigaciones. El presente trabajo amplía y complementa el trabajo "*Dinámica por ordenador con Matlab*" [1], ampliando los conocimientos necesarios para desarrollar modelos con MUBODYNA.

CAPÍTULO I: MODELADO SISTEMAS MECÁNICOS

1. Introducción

En este capítulo se presenta una introducción al modelado de sistemas mecánicos, así como los detalles a tener en cuenta para modelar el sistema por ordenador. El método que se va a explicar a continuación es una base para la correcta definición del sistema en MUBODYNA.

En el presente documento no se explicará el proceso que se lleva a cabo para obtener la solución del movimiento del sistema. Dicho proceso ya lo tiene implementado el programa MUBODYNA, por lo que para la consulta del proceso se recomienda la consulta de los trabajos de Sergio Villar [1], Shabana [2][3], Nikravesh [4][5], y Paulo Flores [6].

1.1 Sistema Mecánico

Un sistema mecánico es un conjunto de cuerpos que se caracteriza por dos características principales: tiene componentes mecánicos que describen desplazamientos de rotación y traslación, y también está constituido por juntas cinemáticas que imponen restricciones en el movimiento relativo de los cuerpos

En un sistema mecánico, los cuerpos pueden ser considerados rígidos o flexibles. Un cuerpo flexible tiene una estructura elástica. Esto quiere decir que los cuerpos que forman el sistema mecánico son deformables con dinámica interna. Por el contrario, cuando se dice que un cuerpo es rígido, se asume que no es deformable. Esto es una hipótesis para simplificar el problema, dado que, al ser flexible, debería ser modelado con seis grados de libertad más las coordenadas generalizadas necesarias para describir las deformaciones.

Durante todo el trabajo se seguirá la hipótesis de cuerpos rígidos para simplificar el problema y modelado de los sistemas.

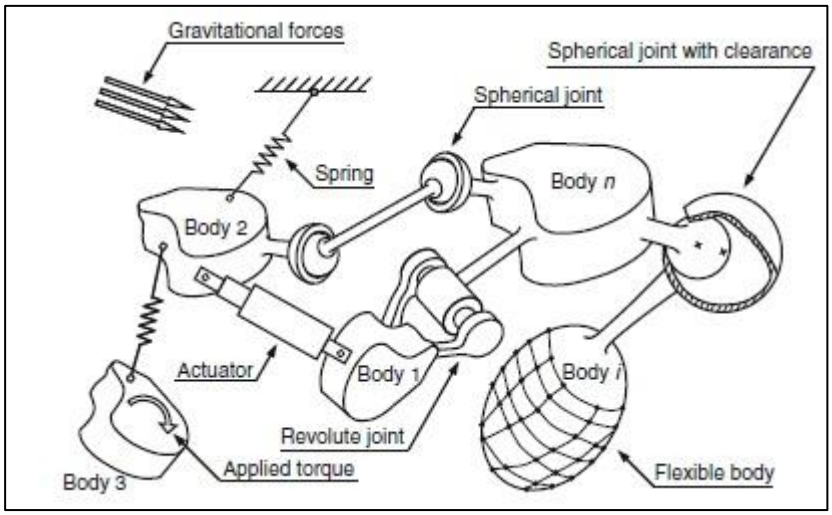

Figura 1. Multiboy System o Sistema Mecánico

En la *Figura 1* se muestra un sistema mecánico con sus elementos principales: cuerpos que forman el sistema, juntas que limitan el sistema, y las fuerzas que se aplican en el sistema mecánico.

Un sistema mecánico puede variar desde un sistema sencillo como un péndulo simple, hasta modelos altamente complejos como la dirección de un automóvil, como el mostrado en la *Figura 2.*

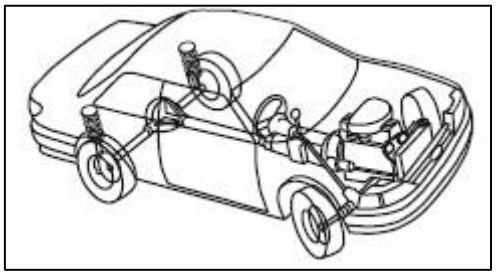

Figura 2. Dirección de un automóvil

2. Conceptos Fundamentales

Antes de comenzar con el modelado de sistemas mecánicos, se van a explicar los conceptos fundamentales para entender el modelado de dichos sistemas, como grados de libertad de un mecanismo, tipos de coordenadas, juntas cinemáticas (aunque se entrará en mayor detalle sobre las juntas en el apartado 3.3).

2.1 Grados de libertad

Para una correcta definición del sistema, son necesarias unas variables con las que sea posible describir, en cualquier instante de tiempo, la posición de cada cuerpo del sistema mecánico.

Se conoce como Número de Grados de Libertad (GDL) al número mínimo de las variables anteriormente mencionadas necesarias para definir la configuración de un mecanismo. Conocer el GDL es el primer paso para el análisis mecánico del sistema, que como se ha mencionado anteriormente no es más que un conjunto de cuerpos unidos mediante juntas cinemáticas.

El GDL se calcula dependiendo de si es un caso tridimensional, o se utiliza un sistema en caso plano.

2.1.1 Caso plano

En el caso plano, el GDL se calcula mediante la siguiente expresión:

$$GDL = 3N - 2p_1 - p_2$$

Siendo:

- N el número de eslabones del mecanismo.
- p_i el número de pares cinemáticos de i grados de libertad.

2.1.2 Caso tridimensional

En el caso tridimensional, el GDL se calcula mediante la siguiente expresión:

$$GDL = 6N - 5p_1 - 4p_2 - 3p_3 - 2p_4 - p_5$$

Siendo:

- N el número de eslabones del mecanismo.
- p_i el número de pares cinemáticos de i grados de libertad.

A continuación, se muestran dos ejemplos del cálculo del número de grados de libertad de un mecanismo.

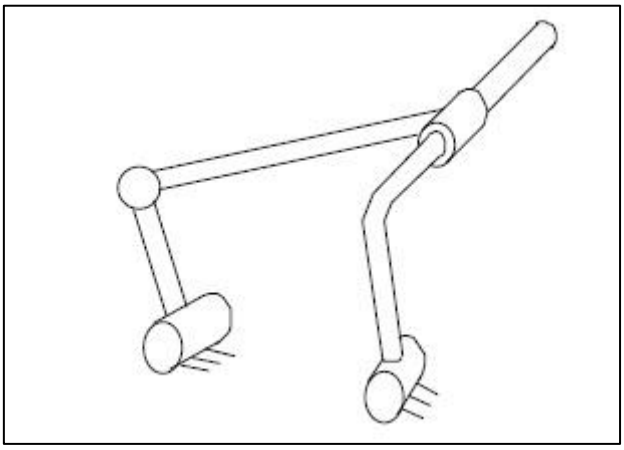

Figura 3. Mecanismo RSCR

La *Figura 3* muestra un mecanismo RSCR. El mecanismo RSCR tiene ese nombre debido a sus 4 pares cinemáticos, 2 pares de revolución, uno esférico, y una cilíndrico.

En este caso, los pares de revolución son pares cinemáticos de 1 grado de libertad, ya que solo permiten el movimiento relativo en una dirección. Por el mismo razonamiento, el par esférico es de 3 grados de libertad, y el cilíndrico de 2 grados de libertad. Por ello, aplicando la ecuación anterior concluimos que el Mecanismo RSCR tiene 1 grado de libertad.

En la Figura *4* se muestra el mecanismo 3 barras, formado por tres barras como su nombre indica, unidas mediante dos pares de revolución y dos pares esféricos. Siguiendo el mismo procedimiento que en el Mecanismo RSCR, el par de revolución es de 1 grado de libertad, y el esférico de 3 grados de libertad. Por ello, y aplicando la ecuación para determinar el GDL, el Mecanismo 4 barras es un mecanismo con 2 grados de libertad.

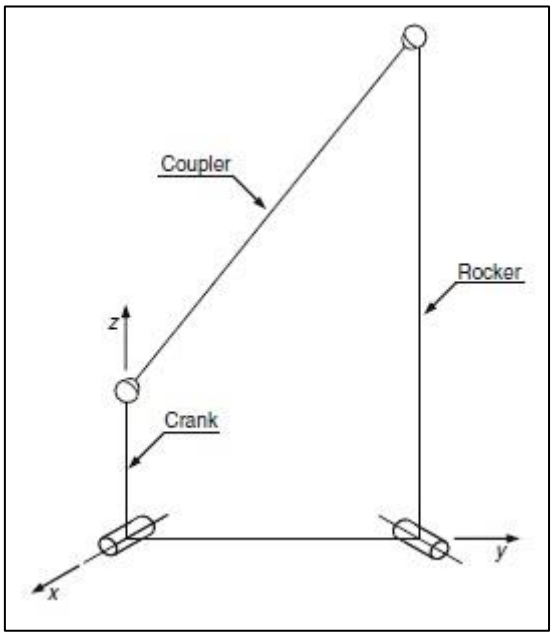

Figura 4. Mecanismo 3 barras

Por último, hay que mencionar que significa el número de grados de libertad de un mecanismo. Si tenemos un mecanismo de 0 GDL, quiere decir que es una estructura, que es un mecanismo inamovible. Si obtenemos un GDL mayor que 0, significa que es un mecanismo que se puede resolver. Cuanto mayor sea el GDL, más variables y ecuaciones de restricción necesitaremos para resolver el mecanismo.

Se puede dar también que obtengamos un GDL menor que 0. Esto quiere decir que representa un mecanismo que no se puede resolver.

2.2 Coordenadas

Como se ha mencionado en el apartado 2.1, para describir la posición de cada cuerpo del sistema en cualquier instante de tiempo, son necesarias unas variables, llamadas coordenadas generalizadas.

Hay varios tipos de criterios para catalogar los diferentes tipos de coordenadas. Un método común para clasificar las coordenadas consiste en dividirlas en coordenadas independientes y coordenadas dependientes. La *Figura 5* muestra la clasificación de los tipos de coordenadas.

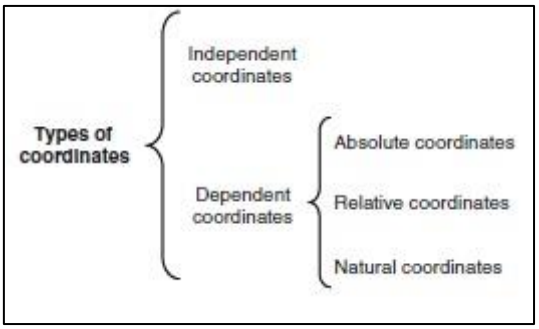

Figura 5. Tipos de coordenadas

Elegir un tipo de coordenada u otra depende del problema a analizar. Una comparación entre diferentes tipos de coordenadas ha sido presentada por Shabana [2], Nikravesh [4] y Jalón Bayo [7], donde se exponen los pros y contras de cada tipo de coordenada.

2.2.1 Coordenadas independientes

Como primera opción están las coordenadas independientes. Modelar un mecanismo en coordenadas independientes significa emplear tantos parámetros como grados de libertad tiene el mecanismo. Esto quiere decir emplear el número mínimo de coordenadas posibles.

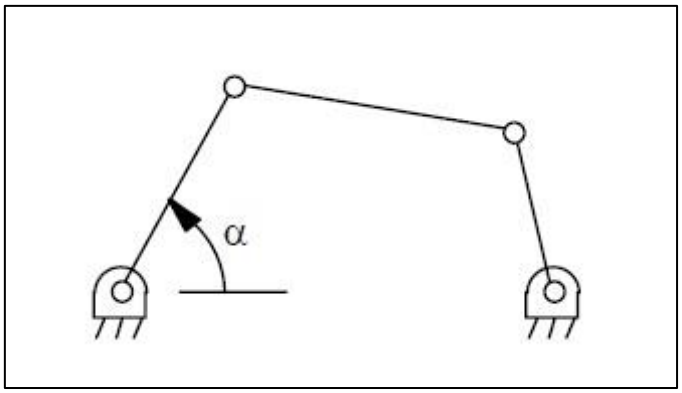

Figura 6. Coordenadas independientes

La *Figura 6* muestra un mecanismo con un grado de libertad. Por tanto, si utilizamos coordenadas independientes solamente necesitaría un único parámetro para definir su movimiento. Este puede ser, por ejemplo, el ángulo que forma la barra izquierda con la horizontal. Sin embargo, para cadenas cerradas, como la mostrada en el ejemplo, no son las coordenadas más apropiadas, dado que exige la resolución de un problema cinemático complicado para determinar la posición del sistema.

Estas coordenadas tienen la principal ventaja de reducir al mínimo el número de coordenadas utilizadas, ya que el número de coordenadas determina el tamaño final del problema.

2.2.2 Coordenadas dependientes

Por otro lado, las coordenadas dependientes son requeridas para satisfacer las ecuaciones de restricción. Esto quiere decir que las coordenadas son dependientes si su número es mayor que el número de grados de libertad del mecanismo.

Como se ha mencionada, estas coordenadas satisfacen las ecuaciones de restricción. Al contar con más variables que grados de libertad, deben existir unas relaciones que las liguen. Estas son las ecuaciones de restricción.

El número de ecuaciones de restricción es la diferencia entre el número de coordenadas dependientes utilizadas y el número de grados de libertad del mecanismo. Las coordenadas dependientes se dividen a su vez en: coordenadas relativas, coordenadas punto referencia y coordenadas naturales.

2.2.2.1 Coordenadas relativas

Las coordenadas relativas sitúan cada elemento del sistema con respecto del anterior en la cadena cinemática. Por tanto, están relacionadas con los pares cinemáticos del mecanismo. Se debe utilizar una por cada grado de libertad que deja un par cinemático.

Las ecuaciones de restricción que ligan las coordenadas proceden del cierre de lazos. En cadenas abiertas serían coordenadas independientes.

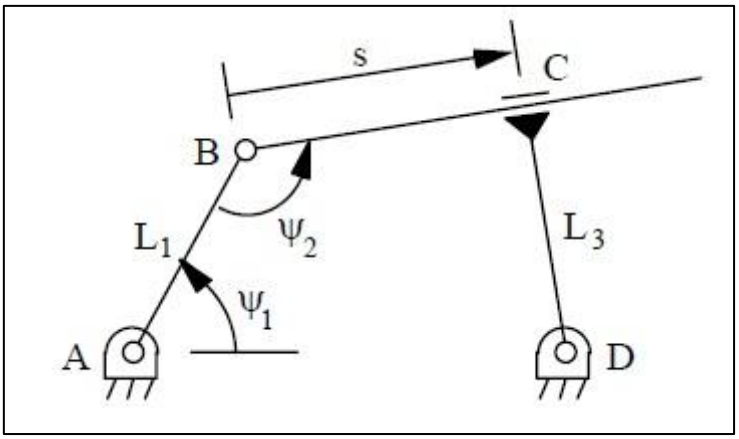

Figura 7. Coordenadas relativas

Por lo tanto, en el ejemplo de la *Figura 7*, la ecuación del cierre de lazo es la siguiente:

$$\overrightarrow{AB} + \overrightarrow{BC} + \overrightarrow{CD} + \overrightarrow{DA} = 0$$

En este caso, el sistema tiene un grado de libertad, y como se observa en la *Figura 7*, se emplean 3 coordenadas relativas (ψ_1, ψ_2, s), por lo que serán necesarias 2 ecuaciones de restricción. De la ecuación de cierre de lazo provienen las 2 ecuaciones de restricción de la siguiente manera:

$$\overrightarrow{AB} + \overrightarrow{BC} + \overrightarrow{CD} + \overrightarrow{DA} = 0 \rightarrow \begin{cases} L_1 \cos\psi_1 + s\cos(\psi_2 + \psi_1 - \pi) + \mathrm{L}_3 \sin(\psi_1 + \psi_2 - \pi) - \mathrm{L}_4 \\ L_1 \sin\psi_1 + s\sin(\psi_2 + \psi_1 - \pi) - \mathrm{L}_3 \cos(\psi_1 + \psi_2 - \pi) - \mathrm{L}_4 \end{cases}$$

Una de las principales ventajas de las coordenadas relativas es su reducido número, que como se ha comentado anteriormente, reducen la dificultad del problema. En el caso de cadena abierta serían coordenadas independientes, dado que no existirían ecuaciones de restricción que las liguen.

Uno de los inconvenientes viene por la aparición de ecuaciones trigonométricas, por lo que la resolución sería más complicada.

2.2.2.2 Coordenadas de punto de referencia

Estas coordenadas se utilizan en cualquier punto del eslabón con la orientación de este, es decir, sitúan a cada elemento independientemente del anterior. En el caso plano la orientación se define mediante un ángulo, y en el caso tridimensional se empleará una de las técnicas conocidas. En el apartado 3.2 se explicará uno de los métodos: Los ángulos de Euler y los parámetros de Euler.

En este caso, las ecuaciones de restricción surgen de imponer las uniones entre los diferentes cuerpos, mediante los pares cinemáticos.

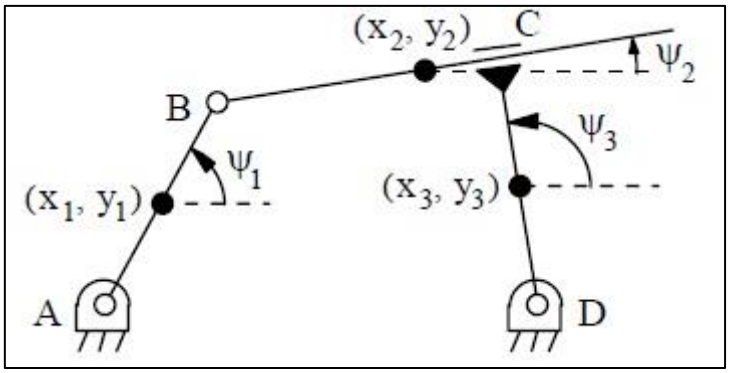

Figura 8. Coordenadas de punto de referencia

La *Figura 8* muestra un ejemplo de modelización con coordenadas de punto de referencia. Una vez definidos los elementos y su orientación, se obtienen 9 coordenadas de punto de referencia, por lo que serán necesarias 8 ecuaciones de restricción. Estas ecuaciones por tanto se establecen en los pares cinemáticos. Las ecuaciones se muestran a continuación:

$$(x_1 - x_A) - \frac{L_1}{2}\cos\psi_1 = 0 \tag{1}$$

$$(y_1 - y_A) - \frac{L_1}{2}\sin\psi_1 = 0 \tag{2}$$

$$\left(x_1 + \frac{L_1}{2}\cos\psi_1\right) - \left(x_2 - \frac{L_2}{2}\cos\psi_2\right) = 0 \tag{3}$$

$$\left(y_1 + \frac{L_1}{2}\cos\psi_1\right) - \left(y_2 - \frac{L_2}{2}\sin\psi_2\right) = 0 \tag{4}$$

$$\psi_3 - (\psi_2 + \frac{\pi}{2}) = 0 \tag{5}$$

$$(x_2 - x_3)\cos\psi_3 + (y_2 - y_3)\sin\psi_3 - \frac{L_3}{2} = 0 \tag{6}$$

$$(x_3 - x_D) - \frac{L_3}{2}\cos\psi_3 = 0 \tag{7}$$

$$(y_3 - y_D) - \frac{L_3}{2}\sin\psi_3 = 0 \tag{8}$$

Las ecuaciones (1) y (2) provienen del par de rotación en A. Las ecuaciones (3) y (4) se corresponden con el par de rotación B. Las ecuaciones (5) y (6) se corresponden con el par prismático C, y las ecuaciones (7) y (8) con el par de rotación en D.

La principal ventaja de este tipo de coordenadas es la fácil definición de las coordenadas y el establecimiento de las ecuaciones de restricción. Por otro lado, como inconveniente se tiene el alto número de coordenadas necesarias, por lo que llevaría a un problema más grande que, por ejemplo, si se emplean coordenadas relativas. Además, en el caso tridimensional la interpretación de la orientación es de elevada complejidad.

2.2.2.3 Coordenadas naturales

Las coordenadas naturales son una evolución de las coordenadas de punto de referencia, explicadas en el apartado 2.2.2.2. Al igual que las coordenadas de punto de referencia, sitúan cada elemento con independencia del anterior. Como diferencia a las anteriores, estas coordenadas se sitúan en los pares cinemáticos, por lo que no son necesarias variables de tipo angular para la definición del sistema.

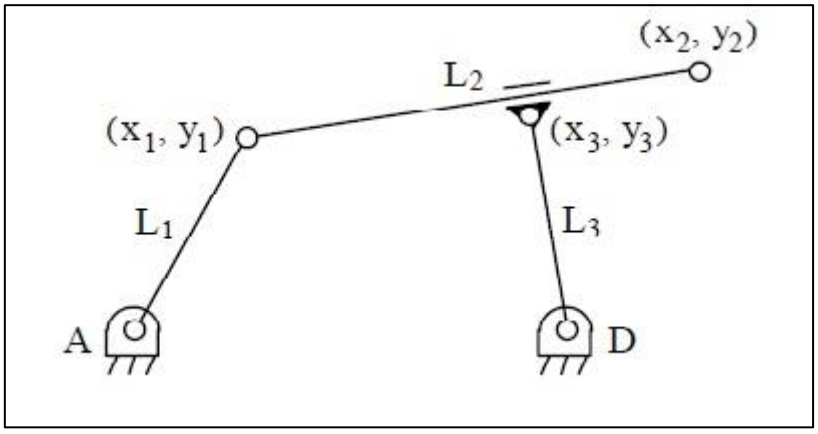

Figura 9. Coordenadas naturales

La *Figura 9* muestra el mismo ejemplo, pero modelado con coordenadas naturales. Al haberse utilizado 6 variables, son necesarias 5 ecuaciones de restricción. Las ecuaciones de restricción son las siguientes:

$$(x_1 - x_A)^2 + (y_1 - y_A)^2 - {L_1}^2 = 0 \tag{9}$$

$$(x_2 - x_1)^2 + (y_1 - y_A)^2 - {L_2}^2 = 0 \tag{10}$$

$$(x_3 - x_D)^2 + (y_3 - y_D)^2 - {L_3}^2 = 0 \tag{11}$$

$$(x_2 - x_1)(x_3 - x_D) + (y_2 - y_1)(y_3 - y_D) = 0 \tag{12}$$

$$(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1) = 0 \tag{13}$$

Las ecuaciones (9), (10) y (11) aseguran la rigidez de los elementos, mediante la condición de distancia constante entre los puntos de las ecuaciones. La ecuación (12) establece el ángulo constante entre las barras, mediante un producto escalar igual a 0. Por último, la ecuación (13) establece el alineamiento de los 3 puntos mediante un producto vectorial entre los vectores 12 y 13.

La ventaja de este tipo de coordenadas está en la sencillez de su definición, por ejemplo, al no aparecer ecuaciones trigonométricas. Por otro lado, como inconveniente de su uso aparece la necesidad de estudiarlas y familiarizarse con ellas previamente.

3. Definición del sistema

En esta sección se va a continuar el estudio del modelado de sistemas mecánicos, introduciendo conceptos a tener en cuenta para el correcto modelado. Como se ha comentado en el apartado 1, este trabajo se va a centrar en la definición y modelado del sistema mecánico. La metodología de sistemas mecánicas también cuenta con una segunda fase, posterior al modelado, de cálculo y desarrollo de modelos matemáticos para obtener la solución del problema. Esta segunda fase se ha explicado en otros manuales de MUBODYNA, y ya está implementada dentro del programa, por lo que no se tratará en apartados siguientes.

3.1 Coordenadas globales y locales

Para un correcto modelado del sistema, se deben emplear un sistema de coordenadas, compuesto por coordenadas globales y coordenadas locales. Antes de continuar, se debe mencionar que durante todo el trabajo se han utilizado sistemas de coordenadas cartesianos.

Las coordenadas globales están representadas por 3 ejes cartesianos, denominados xyz, que parten del mismo punto, denominado el punto origen de todo el sistema mecánico. Sirven para fijar un punto de partida o referencia a la hora de definir la posición de los cuerpos.

Las coordenadas locales, al igual que las globales, están representados por 3 ejes ortogonales, denominados $\xi\eta\zeta$. Las coordenadas locales, aunque no necesariamente, suelen tener su origen en el centro de masas de cada cuerpo. Cada cuerpo tiene su propio sistema de coordenadas locales. Durante el trabajo y en MUBODYNA, las coordenadas locales se emplearán para definir los puntos básicos de los cuerpos.

El sistema de coordenadas local puede trasladarse y rotar respecto al sistema global. Estos parámetros de traslación y rotación deberán definirse para obtener la correcta posición del cuerpo. Puede darse la situación de que el sistema de coordenadas local y global coincidan. En este caso no será necesario definir los parámetros de rotación y traslación, dado que el sistema local ni se traslada ni gira respecto al global.

Por lo tanto, un cuerpo rígido *i* puede ser descrito mediante la definición de un punto arbitrario localizado en el cuerpo, definiendo su posición y la rotación del cuerpo respecto a ese punto.

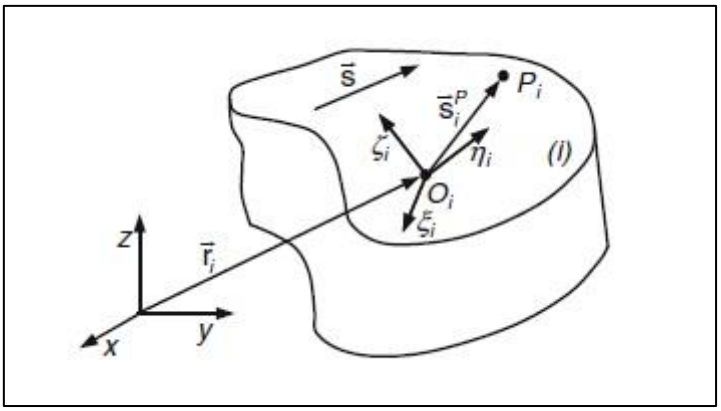

Figura 10. Definición de un punto

La *Figura 10* muestra un ejemplo de definición de un punto P_i del cuerpo *i* respecto a un sistema de referencia global. Se ha utilizado el vector S_i^p para definir el punto P_i respecto al sistema de referencia local. Para definir el origen del sistema local respecto al global se utiliza el vector r_i.

Por lo tanto, la posición del punto P_i respecto al origen del sistema de coordenadas global se expresa de la siguiente manera:

$$r_i{}^p = r_i + s_i{}^P$$

Este es el procedimiento que se sigue en MUBODYNA para definir la posición de un punto de un cuerpo respecto al origen global. Primeramente, se definirá la posición del origen del sistema de referencia local respecto al global, y posteriormente para situar un punto del cuerpo se definirá el punto respecto al sistema local.

Se debe de tener en cuenta que en este apartado no se ha tenido en cuenta una posible orientación del sistema local respecto al global. La manera de definir esta rotación se va a explicar en el siguiente apartado.

3.2 Ángulos de Euler y parámetros de Euler

En el apartado 3.1 se ha explicado los conceptos para definir la posición de los puntos de un cuerpo respecto al sistema de coordenadas global. Para dar una correcta definición de los cuerpos, se debe definir la orientación, como se ha explicado en apartados anteriores. Para ello se van a emplear los ángulos de Euler.

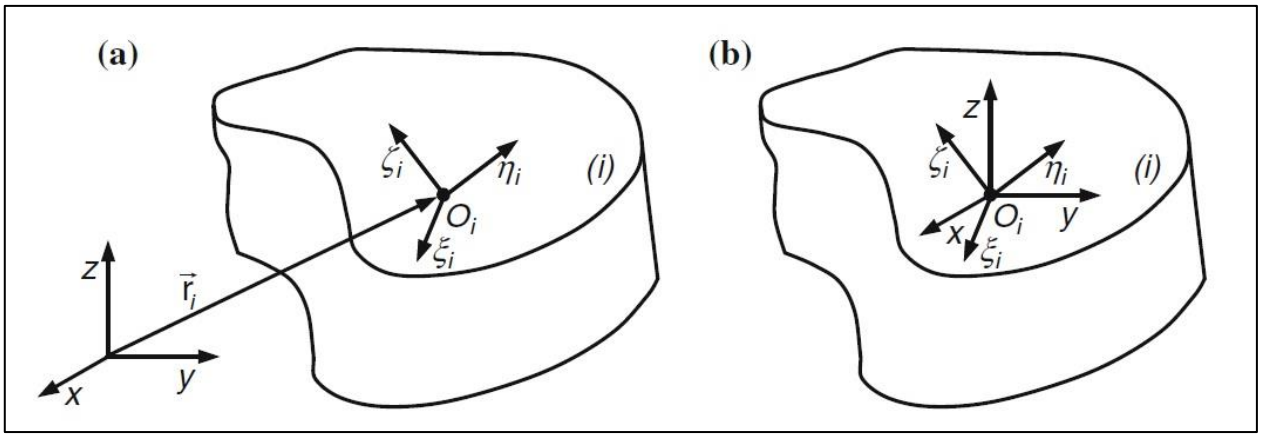

Figura 11. (a) Traslación y rotación (b) Solo rotación

La *Figura 11* muestra dos casos para definir un cuerpo. En el caso (a) se tiene traslación y rotación del sistema local de coordenadas respecto al sistema global xyz. En el caso (b) la posición del origen del sistema global y local coinciden, por lo que la traslación no existe, únicamente existe rotación.

Para una mayor sencillez en la explicación se va a asumir el caso (b), por lo que durante todo este apartado no se considerará la traslación.

La orientación de un cuerpo *i* puede ser definida mediante una matriz A, que es una matriz 3x3. En el apartado 3.1 se ha explicado mediante el ejemplo de la *Figura 10* como definir la posición de un cuerpo mediante el sistema local de coordenadas. Sin embargo, en la *Figura 10* se puede apreciar una rotación del sistema global además de la traslación. Para definir la rotación del sistema se utilizará el vector $s_i'^P$ que contiene, además de la traslación definida en el apartado 3.1 mediante el vector s_i^P, la rotación del sistema local respecto al global. La relación entre estos dos vectores se puede expresar mediante la siguiente expresión: (13)

$$s_i^P = A_i s'^P_i$$

En la ecuación (13) A_i es una matriz de rotación 3x3 que describe la orientación del sistema local respecto al sistema global de coordenadas. Para definir esta matriz, y por tanto la rotación, se van a emplear los ángulos de Euler.

Los ángulos de Euler constituyen un conjunto de tres ángulos que sirven para determinar la rotación de un sistema de referencia (local) respecto a otro sistema de

referencia normalmente fijo (global). Estos ángulos (ψ, θ, σ) representan tres rotaciones consecutivas (respecto a zxz) que mueven el sistema local respecto al global, y con los que quedaría completamente definida la rotación del sistema local. Estas rotaciones sucesivas quedan reflejadas en la matriz A mencionada anteriormente de la siguiente manera:

$$A = DCB$$

donde D, C y B son las matrices de rotación respecto a cada uno de los ejes, que se expresan a continuación:

$$D = \begin{bmatrix} cos\psi & -sen\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sen\theta \\ 0 & sen\theta & cos\theta \end{bmatrix}$$

$$B = \begin{bmatrix} cos\sigma & -sen\sigma & 0 \\ sen\sigma & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Como se aprecia en las matrices, el primer giro de ángulo ψ se realiza respecto al eje z, el segundo de ángulo θ se realiza respecto al eje x, y el tercero de ángulo σ respecto al eje z girado en la segunda rotación.

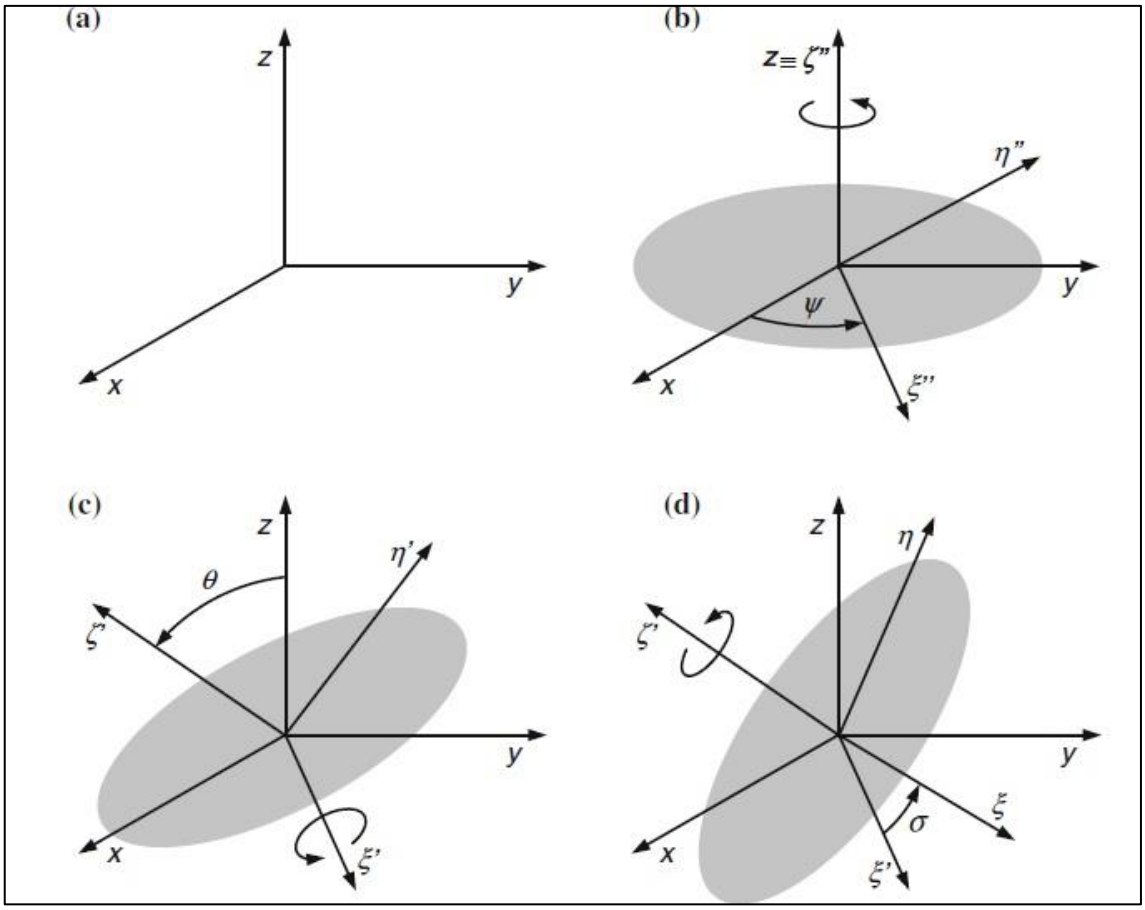

Figura 12. Estados de rotación

La *Figura 12* representa los diferentes estados de rotación respecto a los ejes. El caso (a) muestra el estado inicial, el caso (b) representa la primera rotación, el caso (c) la segunda rotación, y el (d) la tercera y última rotación.

Por otro lado, por el Teorema de Euler de rotación finita, una rotación en tres dimensiones puede ser descrita por un determinado eje y de un determinado ángulo. Este eje imaginario se denomina *u*, y se muestra en la *Figura 13*.

Por lo tanto, un conjunto de coordenadas se puede definir como:

$$e_0 = \cos\frac{\phi}{2}$$

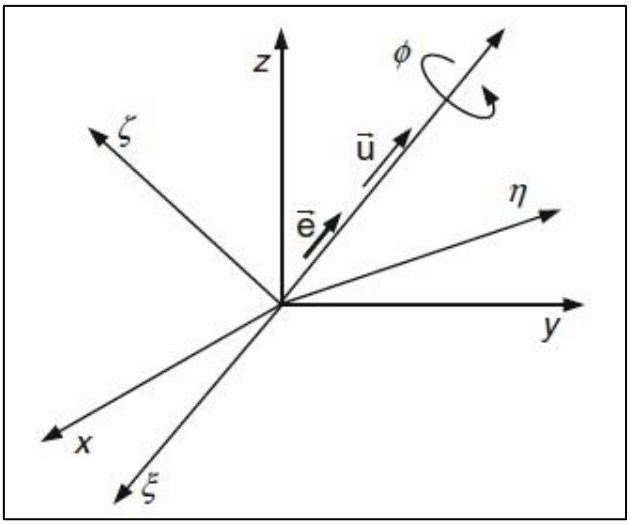

Figura 13. Eje de rotación

En la *Figura 13*, el vector *e* está definido a lo largo del eje de rotación, y tiene una magnitud de sen(ϕ/2):

$$e = \{e_1\ e_2\ e_3\}^T = u \sin\frac{\phi}{2}$$

donde e_0, e_1, e_2 y e_3 son los parámetros de Euler. La suma del cuadrado de cada uno de estos parámetros debe ser igual a 1. Los parámetros de Euler pueden ser expresados de la siguiente manera:

$$p = \begin{Bmatrix} e_0 \\ e \end{Bmatrix}$$

A continuación, se van a mostrar los casos más sencillos de orientación de un sistema local respecto al global. Este caso se da cuando el sistema local gira sobre uno de sus ejes.

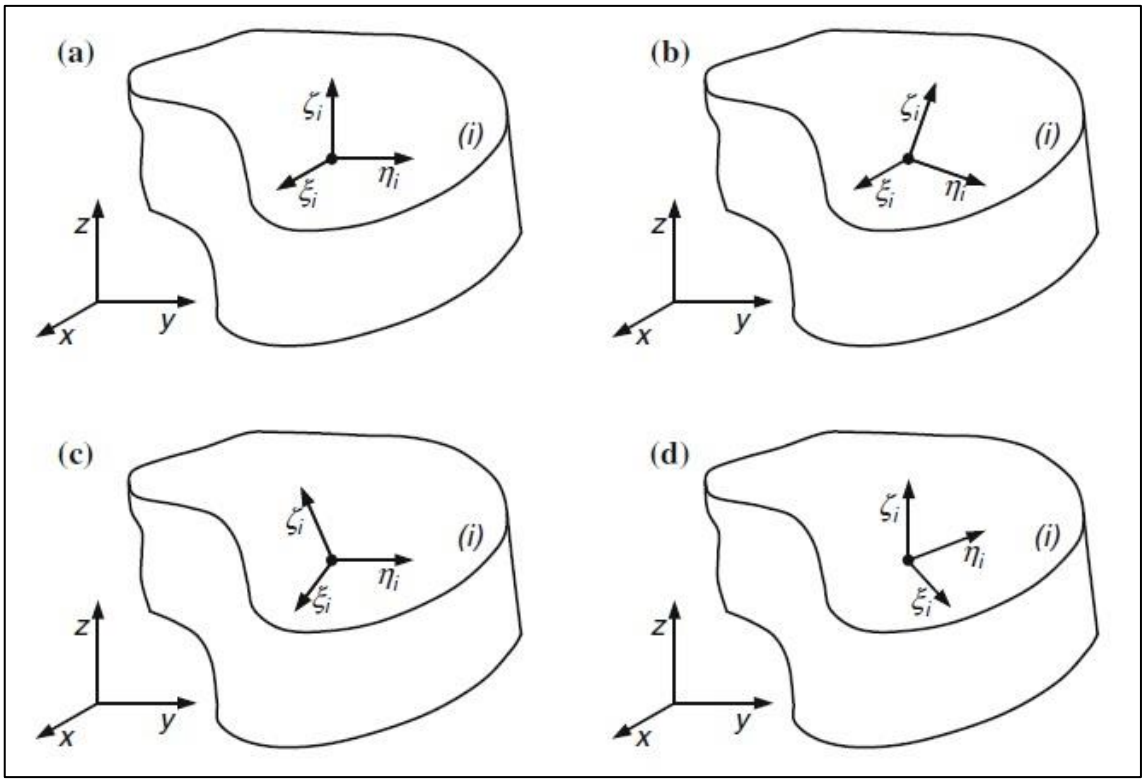

Figura 14. Casos especiales en la rotación que simplifican los parámetros de Euler

La *Figura 14* muestra los casos especiales en los que se simplifican los parámetros de Euler. El caso (a) muestra el caso en el que no existe rotación respecto al sistema global, es decir, el sistema ξηζ es paralelo al xyz. Ente este caso los parámetros de Euler quedan definidos de la siguiente manera:

$$p = [1 \;\; 0 \;\; 0 \;\; 0]^T$$

Cabe mencionar que en MUBODYNA este caso se da por defecto, por lo que no sería necesario definirlos.

En el caso (b) existe rotación de un ángulo ϕ respecto al eje ξ, es decir, es paralelo al eje x. En este caso los parámetros de Euler quedan definidos de la siguiente manera:

$$p = \left[\cos\frac{\phi}{2} \;\; \sin\frac{\phi}{2} \;\; 0 \;\; 0\right]^T$$

En el caso (c) existe rotación de un ángulo ϕ respecto al eje η, es decir, es paralelo al eje y. En este caso los parámetros de Euler quedan definidos de la siguiente manera:

$$p = \left[\cos\frac{\phi}{2} \;\; 0 \;\; \sin\frac{\phi}{2} \;\; 0\right]^T$$

En el caso (d) existe rotación de un ángulo ϕ respecto al eje ζ, es decir, es paralelo al eje z. En este caso los parámetros de Euler quedan definidos de la siguiente manera:

$$p = \left[\cos\frac{\phi}{2} \;\; 0 \;\; 0 \;\; \sin\frac{\phi}{2}\right]^T$$

Este es el método que se va a emplear en MUBODYNA. Primeramente, se definirá la posición del origen del sistema local, y posteriormente la rotación mediante los parámetros de Euler. Una vez definida la posición del sistema local respecto al global, se podrán definir los puntos necesarios para el desarrollo del programa, en coordenadas locales.

3.3 Juntas cinemáticas

Una vez se tienen definidos los diferentes cuerpos que componen el sistema, se debe establecer las relaciones que existen entre ellos. Estas relaciones son las juntas cinemáticas, y en este apartado se van a explicar algunos ejemplos de las juntas más típicas que pueden aparecer en el modelado de sistemas mecánicos.

3.3.1 Junta esférica

La junta esférica restringe los movimientos relativos entre dos cuerpos *i* y *j*, permitiendo 3 rotaciones relativas entre los cuerpos. En el centro de la junta esférica convergen los dos cuerpos, por lo que la condición necesaria a la hora de definir esta junta será imponer que un punto de cada cuerpo permanezca siempre en el centro de la junta.

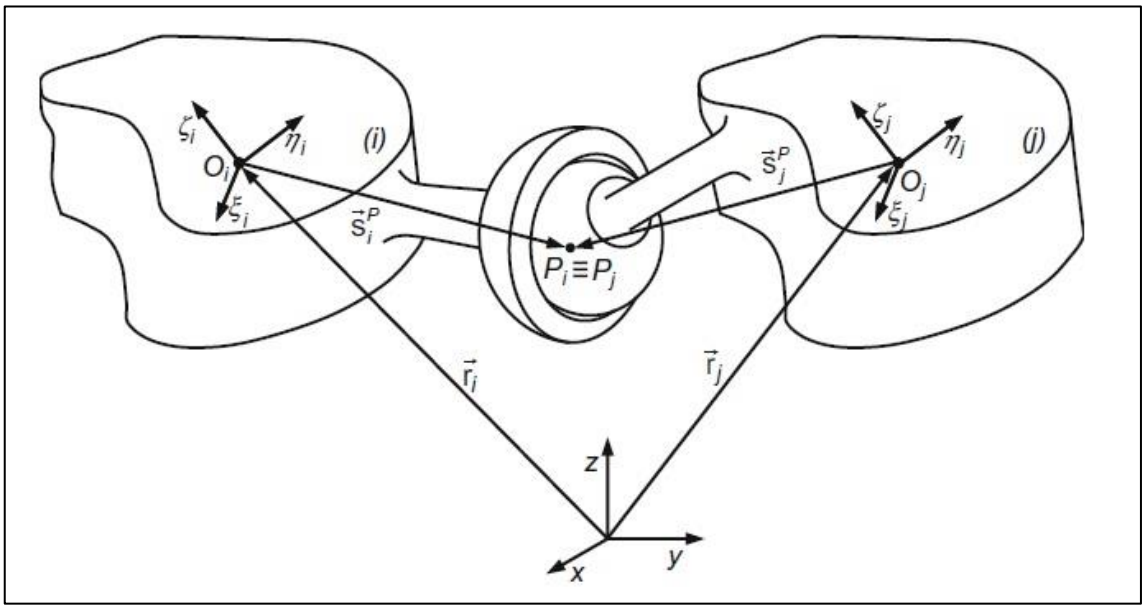

Figura 15. Junta esférica

La *Figura 15* muestra un ejemplo de junta esférica. Por lo tanto, para imponer la condición de junta esférica, se deberá definir, tras haber definido los cuerpos, los puntos

P_i y P_j pertenecientes a los cuerpos i y j respectivamente, en el centro de la junta esférica, y se deberá imponer que estos dos puntos permanezcan siempre en el centro de la junta esférica.

3.3.2 Junta de revolución

La junta de revolución, al igual que la esférica, restringe el movimiento entre dos cuerpos. Esta junta únicamente permite la rotación relativa entre los dos cuerpos alrededor de un eje común a los dos cuerpos.

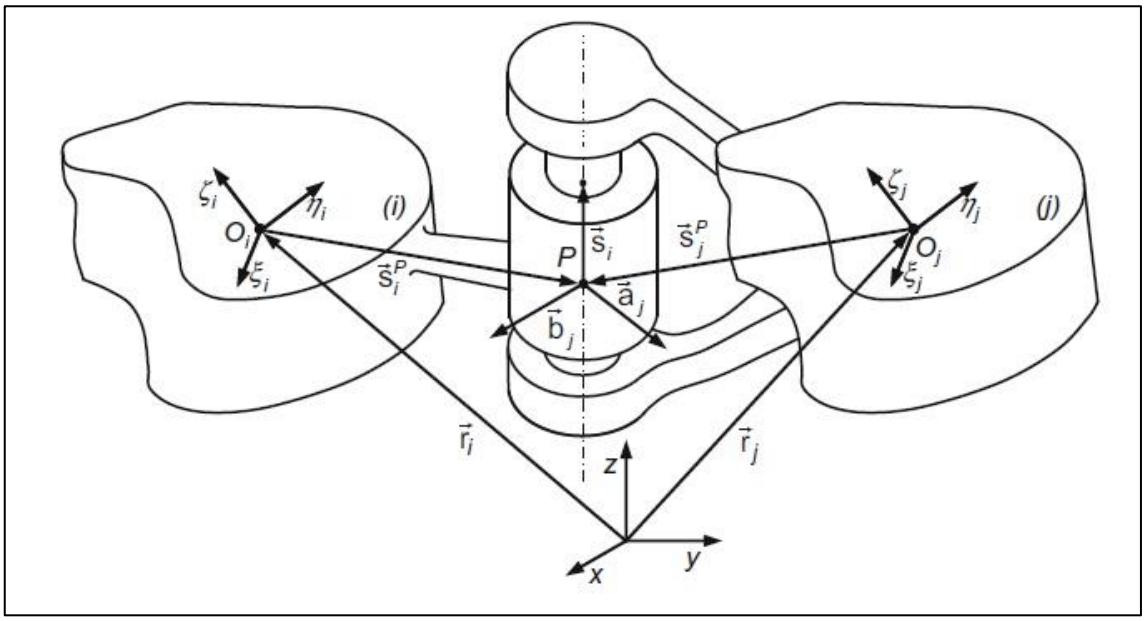

Figura 16. Junta de revolución

De manera similar a la junta esférica, se deberá definir un punto en cada cuerpo que permanecerán siempre en la junta. Además, se deberá imponer una condición que restrinja movimientos para solo permitir la rotación alrededor del eje de revolución. Esta condición puede ser, por ejemplo, definir un vector en cada cuerpo y establecer la condición de que permanezcan perpendiculares en todo momento, como se muestra en la *Figura 16*.

En MUBODYNA se definirán dos vectores paralelos al eje de revolución, uno perteneciente a cada cuerpo, y se establecerá la condición de que permanezcan siempre paralelos.

CAPÍTULO II: MODELOS DESARROLLADOS EN MUBODYNA

En este capítulo se expondrán los modelos desarrollados con MUBODYNA. Un manual extenso, donde se explica en detalle las diferentes secciones de archivos se puede encontrar en "*Dinámica por ordenador con Matlab*" [1], donde explica detalladamente los diferentes archivos y sus funcionalidades, así como la manera de cargar y visualizar la simulación. Durante el primer modelo desarrollado en el presente documento, se introducirá de manera breve los diferentes archivos y sus funcionalidades. Adicionalmente se exponen dos modelos más desarrollados con MUBODYNA.

4. Péndulo - Esfera - Pared

El primer modelo desarrollado en el presente trabajo es un sistema compuesto por un péndulo que golpea una esfera que cae desde una altura de 6 metros, y la cual golpea posteriormente a una pared vertical.

inBodies.m

Para comenzar, como se ha explicado en el apartado 3, se comenzará definiendo los cuerpos. La esfera será el cuerpo 1, el suelo el cuerpo 2, aunque no haría falta definirlo, debido a que por defecto el suelo es el cuerpo 0, el péndulo el cuerpo 3, y, por último, por último, la pared vertical es el cuerpo 4.

```
function inBodies
    include_global
    Body = Body_struct;


B1 = Body;
B1.r = [-0.3; 0;3];
B1.mass = 5;
B1.Jp = [2      0       0
         0      2       0
         0      0       2];
B1.radius = 1;
B1.modulus = 290e9;
B1.poisson = 0.3;

B2 = Body;
B2.r = [0; 0; 0];
B2.mass = 100;
B2.Jp = [0.1    0       0
         0      0.1     0
         0      0       0.1];
B2.modulus = 290e9;
B2.poisson = 0.3;
```

Figura 17. Definición cuerpos

```
B3 = Body;
B3.r = [-3.3; 0; 6];
B3.mass = 5;
B3.Jp = [0.1    0       0
         0      0.1     0
         0      0       0.1];
B3.modulus = 290e9;
B3.poisson = 0.3;


B4 = Body;
B4.r = [4; 0; 0];
B4.mass = 100;
B4.Jp = [0.1    0       0
         0      0.1     0
         0      0       0.1];
B4.modulus = 290e9;
B4.poisson = 0.3;


Bodies = [B1; B2; B3; B4];
```

Figura 18. Definición de cuerpos

Con el comando Bi.r se define la posición del origen del sistema local de cada cuerpo. En el caso de la esfera coincidirá con el centro de la esfera que estará a 3 metros de altura, y la pared vertical se encontrará a 4 metros de distancia horizontal.

Cabe destacar que ningún sistema local ha rotado respecto al global por lo que no es necesario definir los parámetros de Euler, ya que por defecto el sistema local no ha rotado.

inPoints.m

Una vez definidos los cuerpos y sus sistemas locales se definirán los puntos necesarios en cada cuerpo para el correcto funcionamiento del sistema.

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 1;
P1.sPp = [0; 0; 0];

P2 = Point;
P2.Bindex = 1;
P2.sPp = [0; 0; 1];

P3 = Point;
P3.Bindex = 1;
P3.sPp = [0; 0; -0.1];

P4 = Point;
P4.Bindex = 2;
P4.sPp = [3; 0; 0];

P5 = Point;
P5.Bindex = 3;
P5.sPp = [3; 0; 0];
```

Figura 19. Definición de puntos

```
P7 = Point;
P7.Bindex = 0;
P7.sPp = [-0.3; 0; 6];

P8 = Point;
P8.Bindex = 4;
P8.sPp = [0; 0; 6];

Points = [P1; P2; P3; P4; P5; P6; P7; P8];
```

Figura 20. Definición de puntos

Hay que mencionar que los puntos se definen sobre el sistema local de cada cuerpo, mediante el comando Pi.sPp. Los puntos 4 y 8, pertenecientes al suelo y a la pared respectivamente, no son necesarios para el funcionamiento del sistema. Únicamente se emplean para una mejor visualización del sistema.

inVectors1.m

En esta sección no se define nada, dado que no existen vectores de este tipo en este sistema.

inVectors2.m

En esta sección no se define nada, dado que no existen vectores de este tipo en este sistema.

inJoints.m

En esta sección se definirán los pares o juntas cinemáticas. En este caso únicamente se tendrá una junta esférica. Como se ha explicado en el apartado 3.3.1 serán necesarios dos puntos, uno perteneciente a cada cuerpo.

```
function inJoints
    include_global
    Joint = Joint_struct;
J1 = Joint;
J1.type = 'fix';
J1.iBindex = 2;

J2 = Joint;
J2.type = 'sph';
J2.iPindex = 5;
J2.jPindex = 7;

J3 = Joint;
J3.type = 'fix';
J3.iBindex = 4;

Joints = [J1; J2; J3];
```

Figura 21. Definición de juntas

La junta esférica (J2) se define con los puntos 5 y 7, que como se muestra en la *Figura 19* y *Figura 20*, pertenecen al péndulo y al suelo y tienen la misma posición respecto al sistema global.

En la *Figura 21* se muestran dos juntas más del tipo 'fix'. Se emplean para fijar un cuerpo y que no se mueva. Estas juntas se emplean en suelos, paredes, techos…

inForces.m

En esta sección se definen las fuerzas que actúan sobre los cuerpos, incluyendo las fuerzas de impacto que aparecen en el sistema. Una fuerza que aparece en todos los sistemas es la gravedad, que será la fuerza 1. La fuerza 2 es la fuerza de impacto entre la esfera y el suelo, la 3 entre esfera y péndulo, y la 4 entre la esfera y la pared vertical.

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';
S1.gravity = 9.81;

S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 1;
S2.jBindex = 2;
S2.plane_normal_vector = [0; 0; 1];
S2.sphere_radius = 1;
S2.restitution = 1;
S2.force_model = 'lankarani';
S2.n_exponent = 1.5;
S2.friction = false;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
```

Figura 22. Definición de fuerzas

```
S3 = Force;
S3.type = 'sph_pln';
S3.iPindex = 1;
S3.iBindex = 1;
S3.jBindex = 3;
S3.plane_normal_vector = [1; 0; 0];
S3.sphere_radius = 1;
S3.restitution = 1;
S3.force_model = 'lankarani';
S3.n_exponent = 1.5;
S3.friction = false;
S4 = Force;
S4.type = 'sph_pln';
S4.iPindex = 1;
S4.iBindex = 1;
S4.jBindex = 4;
S4.plane_normal_vector = [-1; 0; 0];
S4.sphere_radius = 1;
S4.restitution = 1;
S4.force_model = 'lankarani';
S4.n_exponent = 1.5;
S4.friction = false;
S4.fric_model = 'linear';
S4.mu = 0.5;
S4.mus = 0.6;
Forces = [S1; S2; S3; S4];
```

Figura 23. Definición de fuerzas

En el modelo de fuerzas mostrado en las *Figuras 22 y 23* el coeficiente de restitución es igual a 1. Esto quiere decir que no se consideran las pérdidas de energía en los impactos. Si modelamos un sistema de una esfera cayendo verticalmente contra el suelo y el coeficiente de restitución es igual a 1, no habría pérdida de energía en el choque y por lo tanto la esfera permanecería rebotando infinitamente.

inFuncts.m

En esta sección no es necesario definir nada dado que no existen funciones.

inAnimate.m

En esta sección se define las características de visualización del funcionamiento del sistema

```
function inAnimate
    include_global
    Point = Point_struct;

Points_anim = [];



xmin = -5.5;  xmax = 5.5;
ymin = -7.5;  ymax =  20;
zmin = 0;   zmax =  7;



AZ = 0;
EL = 0;
```

Figura 24. inAnimate.m

La *Figura 24* muestra la definición de esta sección. Primeramente, se definen los límites de visualización, es decir, los valores máximos de nuestro sistema de referencia global. Para continuar se define la rotación de la visualización mediante los comandos AZ y EL. La posición (0,0) de estos comandos nos ofrece una visualización sobre el plano xz global, es decir, una vista frontal. El parámetro AZ gira el sistema respecto el eje z, por lo que un valor de AZ=90 nos mostraría el plano yz. El parámetro EL rota el marco de visualización respecto al eje y.

En este ejemplo se visualiza en 2D, pero para ver una visualización en 3D bastaría con modificar estos parámetros.

inSolver.m

Por último, en esta sección se define las condiciones para obtener la resolución del sistema.

```
function inSolver
    include_global



%--------------------------------
    correct_ic = 'y';
    t_final = 10;
    dt = 0.01;


method = 'standard';
```

Figura 25. inSolver.m

Principalmente se define el tiempo final de simulación, en este caso 10 segundos, y el intervalo de tiempo entre cálculos y grabación de resultados, en este caso, 0.01 segundos.

El comando *correct_ic* establece si el programa hará correcciones iniciales o no (yes o no). Estas correcciones comprueban si todos los puntos de las juntas cinemáticas están bien situados, y en caso de que esté activado los corrige, en caso de que el error sea pequeño.

Cuanto mayor sea el tiempo final de simulación, mayor tardará el programa en ejecutarse, y cuanto menor sea el intervalo de tiempo entre cálculos y grabación de datos, más precisa será la simulación, pero más tiempo empleará el programa en determinar la solución.

También se debe destacar el método que se utilice para la solución. El modelo de utilizado en el presente documento es el modelo 'standard'. Este es el modelo de solución más sencillo, pero el también el menos preciso. Al modelar sistemas con un tiempo elevado, el error crece gradualmente con el tiempo, por lo que en la simulación se observan incoherencias en los últimos segundos. Para minimizar el error, existen dos métodos más: 'baumgarte' y 'aumented'. El inconveniente es que el programa necesitará más tiempo para determinar la solución.

Simulación

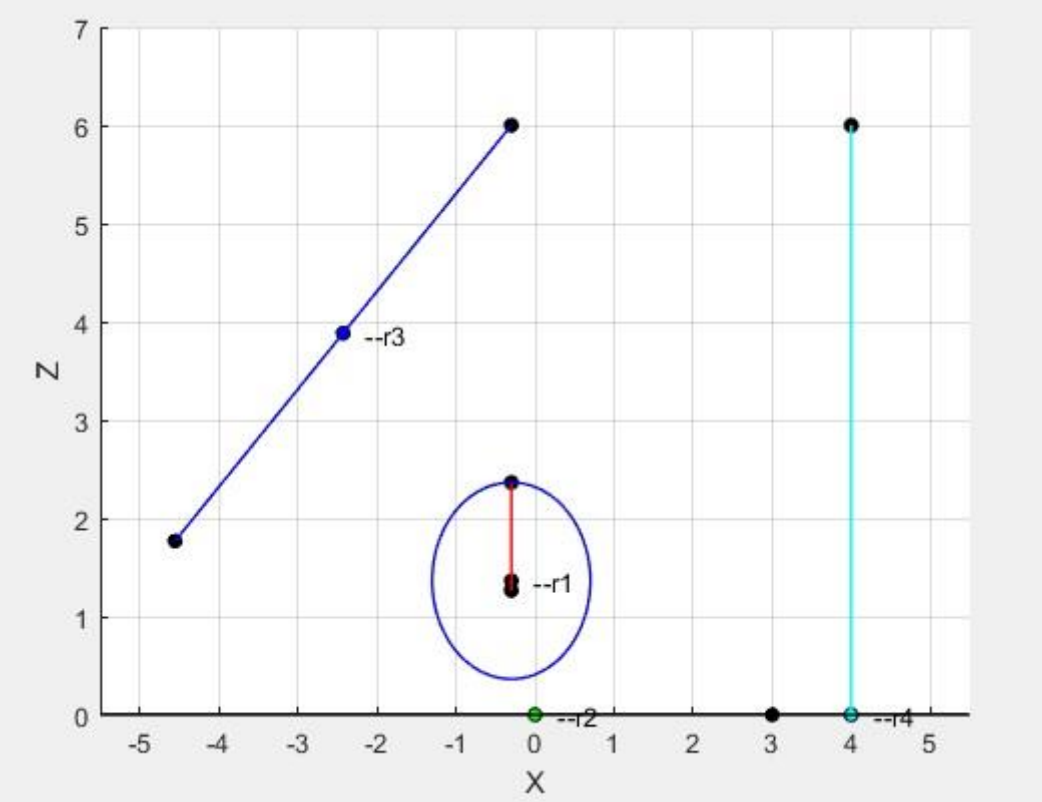

Figura 27. Simulación

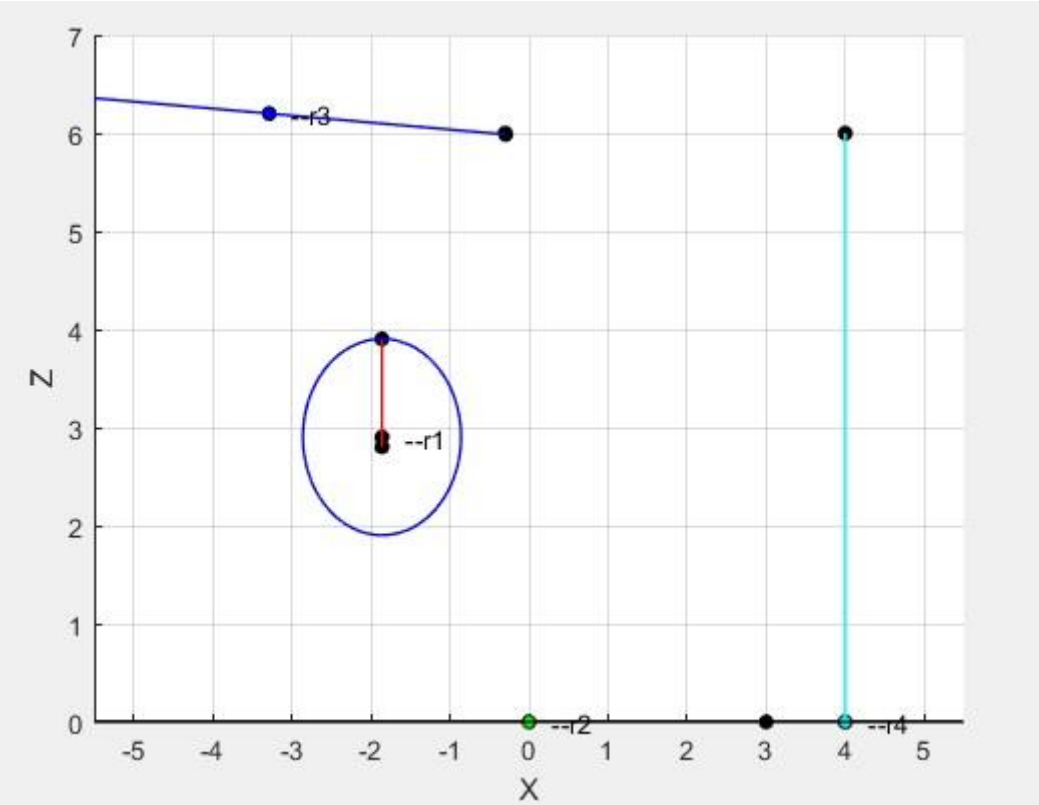

Figura 26. Simulación

En la *Figura 26* se muestra un instante puntual de la simulación del presente. Se puede apreciar como el péndulo, que ha comenzado en posición completamente horizontal y ya se ha desplazado, igual que la esfera, cuyo centro se situaba en el instante inicial a una altura de 3 metros. La pared, como se ha explicado anteriormente, se sitúa a un a distancia de 4 metros horizontalmente del centro de la esfera en el instante inicial.7

En la *Figura 27* se muestra un instante puntual de los momentos finales de la simulación, donde se puede apreciar como la esfera ha rebotado en la pared y ha golpeado el péndulo.

5. Esfera-esfera

Este modelo se representa el contacto entre esferas. Para ello se definen dos esferas cayendo verticalmente desde diferentes alturas, hasta llegar a contactar entre ellas.

inBodies.m

Se definen 3 cuerpos: 2 esferas y 1 plano que actúa como suelo. El suelo no haría falta definirlo ya que está definido por defecto como el cuerpo 0.

```
function inBodies
    include_global
    Body = Body_struct;


B1 = Body;
B1.r = [0.3; 0;6];
B1.mass = 5;
B1.Jp = [2      0       0
         0      2       0
         0      0       2];
B1.radius = 1;
B1.modulus = 290e9;
B1.poisson = 0.3;

B2 = Body;
B2.r = [0; 0; 0];
B2.mass = 100;
B2.Jp = [0.1    0       0
         0      0.1     0
         0      0       0.1];
B2.modulus = 290e9;
B2.poisson = 0.3;
```

Figura 28. Definición cuerpos

```
B3 = Body;
B3.r = [0.3; 0;15];
B3.mass = 5;
B3.Jp = [2      0       0
         0      2       0
         0      0       2];
B3.radius = 1;
B3.modulus = 290e9;
B3.poisson = 0.3;


Bodies = [B1; B2; B3];
```

Figura 29. Definición cuerpos

Las dos esferas son idénticas. El cuerpo 1 es la primera esfera cayendo desde una altura de 6 metros, y el cuerpo 3 la segunda esfera cayendo desde la misma posición en x e y, pero desde una altura de 15 metros. El cuerpo 2 es el suelo.

inPoints.m

En este sistema se definen los centros de las esferas, y dos puntos adicionales para visualizar el diámetro de la esfera. Los puntos de la esfera para dibujar el diámetro son muy útiles en modelos en los que la esfera gira para apreciar la rotación.

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 1;
P1.sPp = [0; 0; 0];

P2 = Point;
P2.Bindex = 1;
P2.sPp = [0; 0; 1];

P3 = Point;
P3.Bindex = 1;
P3.sPp = [0; 0; -1];

P4 = Point;
P4.Bindex = 3;
P4.sPp = [0; 0; 0];
```

Figura 30. Definición de puntos

```
P5 = Point;
P5.Bindex = 3;
P5.sPp = [0; 0; 1];

P6 = Point;
P6.Bindex = 3;
P6.sPp = [0; 0; -1];

Points = [P1; P2; P3; P4; P5; P6];
```

Figura 31. Definición de puntos

Como se muestra en las *Figuras 30 y 31* únicamente se definen los centros de las esferas y dos puntos adicionales en cada esfera para apreciar el diámetro.

inVectors1.m

En esta sección no se define nada, dado que no existen vectores de este tipo en este sistema.

inVectors2.m

En este ejemplo se debe definir un vector de tipo 2 que una los centros de las esferas.

```
function inVectors2
    include_global
    Vector = Vector2_struct;

V1 = Vector;
V1.iPindex = 1;
V1.jPindex = 4;

Vectors2 = [V1];
```

Figura 32. Definición de vectores de tipo 2

inJoints.m

```
function inJoints
    include_global
    Joint = Joint_struct;
J1 = Joint;
J1.type = 'fix';
J1.iBindex = 2;

Joints = [J1];
```

Figura 33. Definición de juntas

En este sistema no existen pares cinemáticos. Sin embargo, al igual que en el ejemplo del péndulo-esfera-pared 4, es necesario definir una junta de tipo ‘fix’ para fijar el cuerpo 2 y que no experimente traslación ni rotación en la simulación, es decir, que no se mueva.

inForces.m

En este caso, existen fuerzas de impacto y el peso.

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';
S1.gravity = 9.81;

S2 = Force;
S2.type = 'sph_pln';
S2.iPindex = 1;
S2.iBindex = 1;
S2.jBindex = 2;
S2.plane_normal_vector = [0; 0; 1];
S2.sphere_radius = 1;
S2.restitution = 1;
S2.force_model = 'lankarani';
S2.n_exponent = 1.5;
S2.friction = false;
S2.fric_model = 'linear';
S2.mu = 0.5;
S2.mus = 0.6;
```

Figura 34. Definición de fuerzas

La fuerza 1 la fuerza de la gravedad, y la fuerza 2 la fuerza de impacto entre la primera esfera y el suelo.

```
S3 = Force;
S3.type = 'sph_pln';
S3.iPindex = 4;
S3.iBindex = 3;
S3.jBindex = 2;
S3.plane_normal_vector = [0; 0; 1];
S3.sphere_radius = 1;
S3.restitution = 1;
S3.force_model = 'lankarani';
S3.n_exponent = 1.5;
S3.friction = false;
S3.fric_model = 'linear';
S3.mu = 0.5;
S3.mus = 0.6;
```

Figura 35. Definición de fuerzas

La fuerza 3 es la fuerza entre de impacto la segunda esfera y el suelo

```
S4 = Force;
S4.type = 'sph_sph';
S4.iPindex = 1;
S4.jPindex = 4;
S4.iBindex = 1;
S4.jBindex = 3;
S4.V2index = 1;
S4.restitution = 1;
S4.force_model = 'lankarani';
S4.friction = false;
S4.fric_model = 'linear';
S4.mu = 0.5;
S4.mus = 0.6;

Forces = [S1; S2; S3; S4];
```

Figura 36. Definición de fuerzas

La fuerza 4 es la fuerza de contacto entre as dos esferas. Como se muestra en la *Figura 36* es una fuerza similar a la fuerza de impacto entre una esfera y un plano. Se deben definir los centros de cada una de las esferas respectivamente, además de un vector de tipo 2 que une los centros de las esferas.

inFuncts.m

En esta sección no es necesario definir nada dado que no existen funciones.

inAnimate.m

```
function inAnimate
    include_global
    Point = Point_struct;

Points_anim = [];




xmin = -20;  xmax = 20;
ymin = -7.5;  ymax =  20;
zmin = 0;  zmax =  20;




AZ = 0;
EL = 0;
```

Figura 37. inAnimate

En este ejemplo se visualiza también en 2D, y concretamente, como se muestra en la *Figura 37* en el plano xz.

inSolver.m

```
function inSolver
    include_global
%-----------------------------------



%-----------------------------------
    correct_ic = 'y';
    t_final = 10;
    dt = 0.01;


method = 'standard';
```

Figura 38. inSolver

En este caso el tiempo final de simulación es de 10 segundos, y el intervalo de tiempos es 0.01. Se ha utilizado el método 'estándar', y realiza la corrección inicial.

Simulación

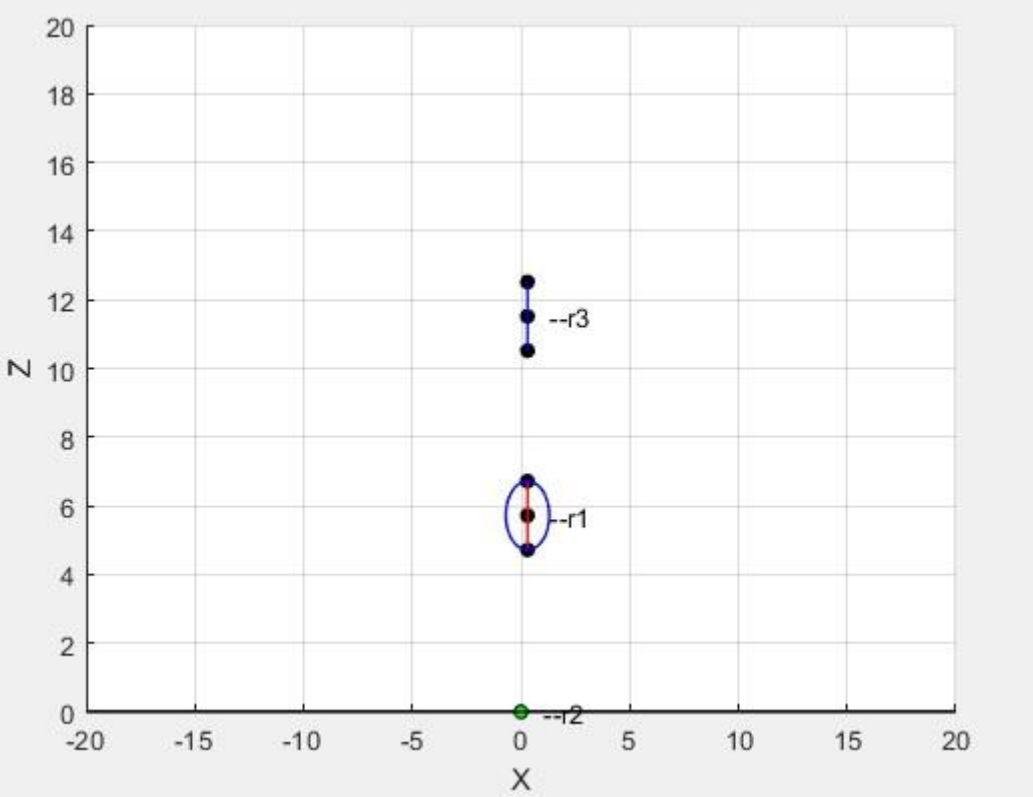

Figura 39. Simulación

En la *Figura 39* se muestra un instante puntual de la simulación del presente modelo. Se puede apreciar como la segunda esfera no se visualiza de manera correcta. Esto se debe a que en el algoritmo no está implementada, y se sigue investigando para poder visualizarla. A pesar del error de visualización, los cálculos y simulación del modelo se han realizado de manera correcta, apreciando claramente el impacto entre las esferas.

6. Deslizadera

Este modelo sistema está formado por 3 barras en el cual una barra está unida al suelo mediante una junta de revolución, la cual rota respecto al eje y, y otra está unida al suelo mediante una deslizadera paralela al eje y. La última barra (cuerpo 3) sirve para simular una junta cilíndrica que permita la rotación respecto a un eje y la traslación respecto a otro cuerpo. Es una combinación de la junta de revolución y traslación.

Se ha implementado un motor en la junta 1 (revolución) para que el sistema gire de manera constante. Para ello se debe definir una función en el archivo inFuncts.m.

Este modelo representa, por ejemplo, el movimiento de un pistón en el motor de un vehículo.

inBodies.m

El sistema contiene 3 cuerpos, por lo que el archivo inBodies.m se definiría de la siguiente manera:

```
function inBodies
    include_global
    Body = Body_struct;

B1 = Body;     % barral
B1.r = [0; 0; 2];

B1.mass = 5;
B1.Jp = [0.005  0           0
         0      0.005      0
         0      0            0.005];

B1.w = [0; 5; 0];

B1.p = [1;0;0;0];
```

Figura 40. inBodies

```
B2 = Body;
B2.r = [0; 1.5; 2.25];
B2.mass = 5;
B2.Jp = [0.005  0           0
       0      0.005      0
        0      0            0.005];
phi = -40.60*pi/180;
    p2 = phi/2;
    e0 = cos(p2);
    e1 = sin(p2);
B2.p = [e0;e1;0;0];

B3 = Body;
B3.r = [0; 3; 0.25];

B3.mass = 0.5;
B3.Jp = [0.005  0           0
         0      0.005      0
         0      0            0.005];

Bodies = [B1; B2; B3];
```

Figura 41. inBodies

En las *Figuras 40 y 41* se muestra la definición de los cuerpos. En el cuerpo 1 se ha definido una velocidad angular en el eje local y, mediante el comando *B1.w.*

El sistema local del cuerpo 2 ha rotado, además de la traslación, respecto al sistema global de coordenadas. Para la rotación, como se ha explicado en el apartado 3.2, se han empleado los parámetros de Euler, siendo el caso (b) de la *Figura 14*. En este caso ha rotado un valor de -40.60 º.

Por último, la barra 3 junto con la junta esférica para conectarlo con el cuerpo 2, que se definirá en el archivo inJoints.m, se emplea para simular la junta cilíndrica.

inPoints.m

```
function inPoints
    include_global
    Point = Point_struct;

P1 = Point;
P1.Bindex = 1;
P1.sPp = [0; 0; -2];

P2 = Point;
P2.Bindex = 1;
P2.sPp = [0; 0; 2];

P3 = Point;
P3.Bindex = 0;
P3.sPp = [0; 0; 0];

P4 = Point;
P4.Bindex = 2;
P4.sPp = [0; -2.305; 0];

P5 = Point;
P5.Bindex = 2;
P5.sPp = [0; 2.305; 0];
```

Figura 42. inPoints

```
P6 = Point;
P6.Bindex = 3;
P6.sPp = [0; 0; 0.25];

P7 = Point;
P7.Bindex = 3;
P7.sPp = [0; 0; -0.25];

P8 = Point;
P8.Bindex = 0;
P8.sPp = [0; 3; 0];

Points = [P1; P2; P3; P4; P5; P6; P7;P8];
```

Figura 43.inPoints

Las *Figuras 42 y 43* muestran el archivo inPoints.m en este sistema. Se han definido los puntos iniciales y finales de cada barra. Se debe prestar atención a la definición de los puntos 4 y 5, dado que pertenecen al cuerpo 2 y el sistema local ha rotado respecto al global, y el comando Pi.sPp define los puntos respecto al sistema local de coordenadas.

También se deben definir los puntos 3 y 8 pertenecientes al suelo (cuerpo 0) para definir los pares cinemáticos que unen el sistema al suelo.

inVectors1.m

```
function inVectors1
    include_global
    Vector = Vector1_struct;

V1 = Vector;
V1.Bindex = 1;
V1.sp = [0; 1; 0];

V2 = Vector;
V2.Bindex = 0;
V2.sp = [0; 1; 0];

V3 = Vector;
V3.Bindex = 0;
V3.sp = [0; 1; 0];

Vectors1 = [V1; V2; V3];
```

Figura 44. inVectors1

En la *Figura 44* se muestra el archivo inVectors1.m. En este sistema son necesarios 3 vectores de tipo 1. Los vectores 1 y 2 se emplean para definir la junta de revolución entre el suelo y el cuerpo 1. Este vector unitario se define en la dirección del eje de revolución de la junta.

Por otro lado, el vector 3 es necesario para definir la junta de traslación (deslizadera) entre el cuerpo 3 y el suelo. En este caso, el deslizamiento se produce en dirección paralela al eje y del sistema global, y como es el cuerpo 3 el que "desliza" sobre el suelo, el vector pertenecerá al suelo y tendrá la dirección de la deslizadera. Es importante definir correctamente este vector, ya que, si se define un movimiento imposible en el sistema, el programa no compilará y se obtendrá un error.

inVectors2.m

```
function inVectors2
    include_global
    Vector = Vector2_struct;
V1 = Vector;
V1.iPindex = 7;
V1.jPindex = 8;

Vectors2 = [V1];
```

Figura 45. inVectors2

En la *Figura 45* se muestra el archivo inVectors2.m. En este ejemplo es necesario un vector de tipo 2 que una los puntos iniciales de la deslizadera. Estos puntos son el punto 7 y 8 pertenecientes al cuerpo 3 y al suelo, respectivamente.

inFuncts.m

Una definición de las funciones posibles en MUBODYNA, así como su correcta definición se da en "*Dinámica por ordenador con Matlab*"[1].

```
function inFuncts
    include_global
    Funct = Funct_struct;

C1 =     Funct;
C1.type = 'a';
C1.coeff = [0 5 0];

Functs = [C1];
```

Figura 46. inFuncts

En la *Figura 46* se muestra el archivo inFuncts.m del presente sistema. La función se define para la implementación del motor en la junta de revolución del cuerpo 1 con el suelo.

La función de tipo a sigue la siguiente expresión:

$$f = C_1 + C_2 x + C_3 x^2$$

Por lo tanto, es necesario definir los coeficientes C_1, C_2 y C_3. En este caso C_1 y C_3 son 0 y C_2 es igual a 5: el valor de la rotación definida en el archivo inBodies.m.

inJoints.m

Este modelo consta de 4 pares cinemáticos, además de una junta que se debe definir debido al motor.

```
function inJoints
    include_global
    Joint = Joint_struct;

J1 = Joint;
J1.type = 'rev';
J1.iPindex = 1;
J1.jPindex = 3;
J1.iVindex_a = 1;
J1.jVindex_a = 2;

J2 = Joint;
J2.type = 'rel-rot';
J2.iBindex = 1;
J2.jVindex_a = 2;
J2.iFunct = 1;

J3 = Joint;
J3.type = 'sph';
J3.iPindex = 2;
J3.jPindex = 4;
```

Figura 47.inJoints

```
J4 = Joint;
J4.type = 'sph';
J4.iPindex = 6;
J4.jPindex = 5;

J5 = Joint;
J5.type = 'tran';
J5.jVindex_a = 3;
J5.V2index = 1;

Joints = [J1; J2; J3; J4; J5];
```

Figura 48.inJoints

La junta 1 es una junta de revolución, que une la barra 1 con el suelo. Para definirla se emplean los dos vectores explicados en el archivo de inVectors1.m, y los puntos en los que se aplican los vectores, es decir, los puntos donde se unen los cuerpos. El eje de revolución se sitúa en el eje y. La junta 2 es la empleada para definir el motor. Para definirla es necesario el vector perteneciente al suelo que se ha empleado para definir la junta de revolución, además la función que define el valor del motor. También es necesario especificar cuál es el cuerpo, en este caso el 1, el que gira gracias a la acción del motor. Las juntas 3 y 4 son juntas esféricas, que unen la barras 1 con la 2, y la 2 con la 3, por lo que serán necesarios, como se ha explicado anteriormente, dos puntos por cada junta. Por último, la junta 4 es la junta de traslación. Se debe indicar los puntos que coinciden inicialmente, mediante un vector de tipo 2, y un vector de tipo 1 que indique la dirección de la traslación o deslizadera. Estos son los vectores de los que se ha hablado en los archivos de inVectors1.m e inVectors2.m

inForces.m

En este sistema solo existe la fuerza debido a la gravedad.

```
function inForces
    include_global
    Force = Force_struct;

S1 = Force;
S1.type = 'weight';
S1.gravity = 9.81;
S1.wgt = [0; 0; -1];

Forces = [S1];
```

Figura 49. inForces

En la *Figura 49* se muestra el archivo inForces.m para el presente sistema.

inAnimate.m

En este sistema, también se visualiza en 2D, aunque también es interesante visualizarlo en 3D, modificando los parámetros AZ y EL. En la *Figura 50* se muestra el archivo inAnimate.m del presente modelo.

```
function inAnimate
    include_global
    Point = Point_struct;

Points_anim = [];



xmin = -5;  xmax =  5;
ymin = -7;  ymax =  10;
zmin = -10;  zmax =  10;



AZ = 90;
EL = 0;
```

Figura 50. inAnimate

inSolver.m

```
function inSolver
    include_global
%-----------------------------------



%-----------------------------------
    correct_ic = 'y';
    t_final = 10;
    dt = 0.01;


method = 'standard';
```

Figura 51. inSolver

En este sistema, el tiempo final de simulación es de 10 segundos, y el intervalo de tiempos es 0.01. Se ha utilizado el método ‘estándar’, y realiza la corrección inicial.

Simulación

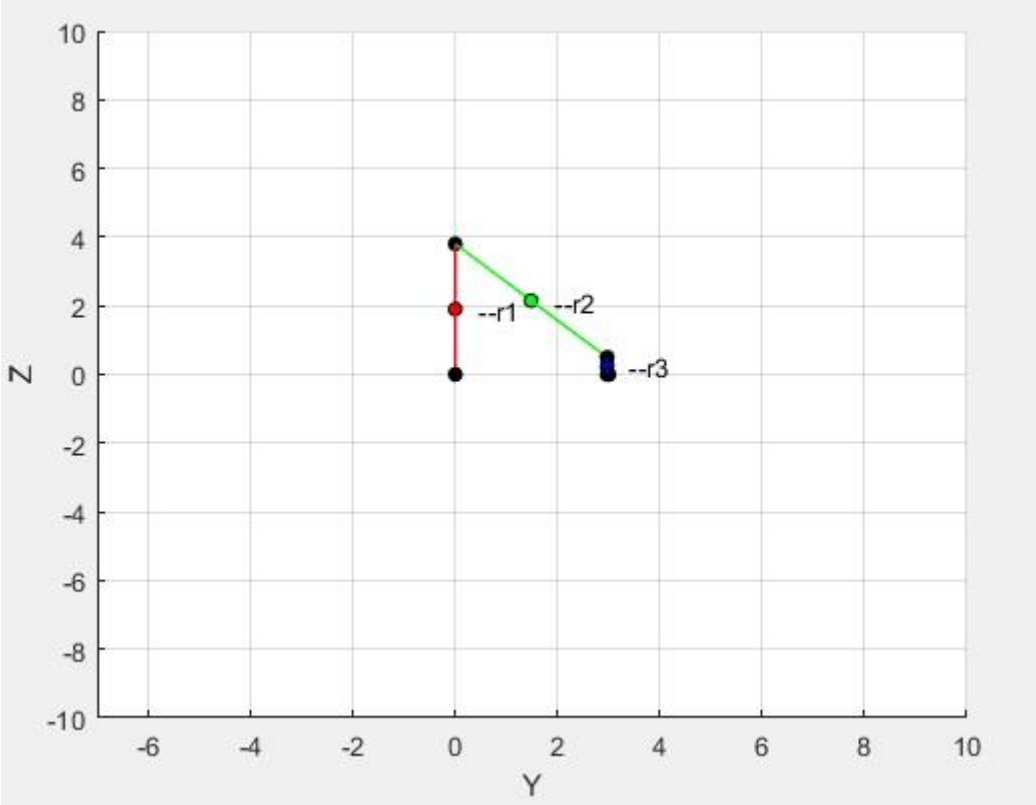

Figura 53. Simulación

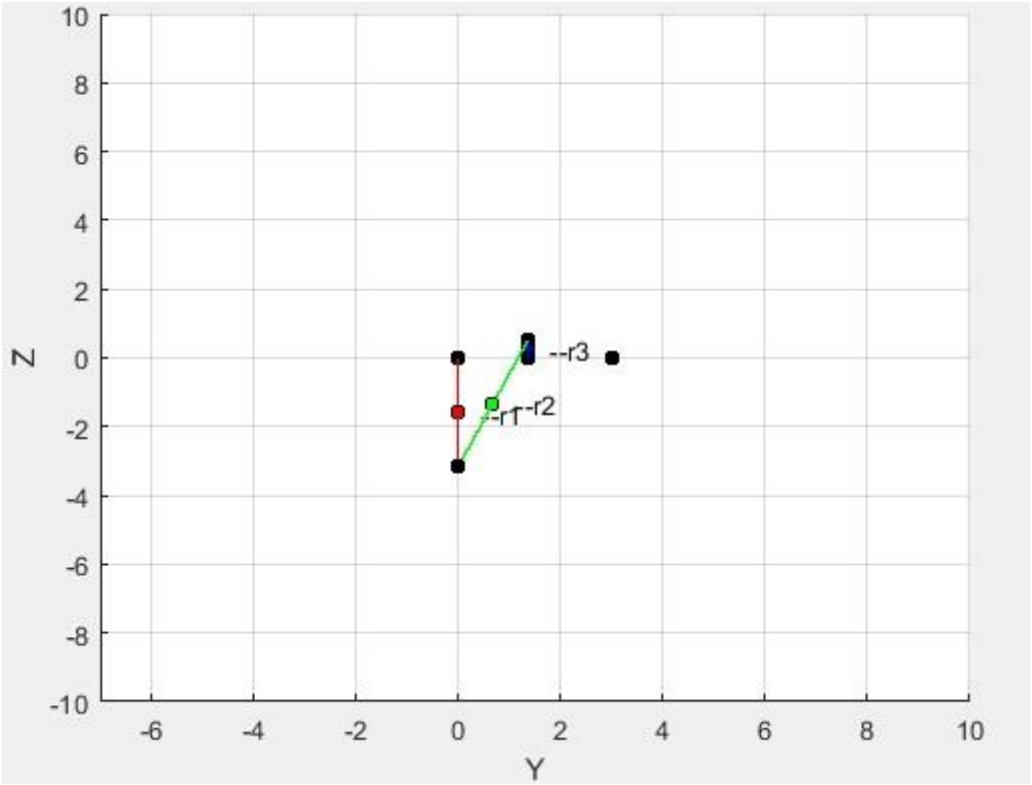

Figura 52. Simulación

En la *Figura 52* se muestra el instante inicial del modelo, y en la *Figura 53* se muestran un instante determinado. Se puede apreciar como el cuerpo 1 alrededor del eje y, y como la deslizadera se traslada a lo largo del eje y. Es interesante también visualizar el modelo en 3D.

7. Comandos save y load

Para finalizar la simulación de los modelos, se va a explicar dos comandos muy útiles a la hora de definir y simular un sistema en MUBODYNA. Se tratan de los comandos de save y load.

Tras finalizar la definición y simulación de un modelo, se puede guardar la simulación del movimiento del sistema, para así evitar volver a calcular y simular todo el archivo del modelo antes de la simulación. Esto es de gran utilidad para modelos cuya simulación tarde en calcularse horas, incluso días.

En modelos sencillos cuya simulación solo tarde unos segundos en calcularse no es útil para este fin, pero si es útil para guardar la simulación y modificar el sistema, sin perder la simulación inicial.

Para guardar una simulación se deberán seguir los siguientes pasos:

- Simular un modelo xxx, como se explica en el trabajo "*Dinámica por ordenador con Matlab"[1]*
- Guardar la simulación mediante el comando save xxx.mat.

Para mostrar la simulación se deben seguir los siguientes pasos:

- Agregar las carpetas de MUBODYNA a la ruta.
- Cargar la simulación mediante el comando load xxx.mat

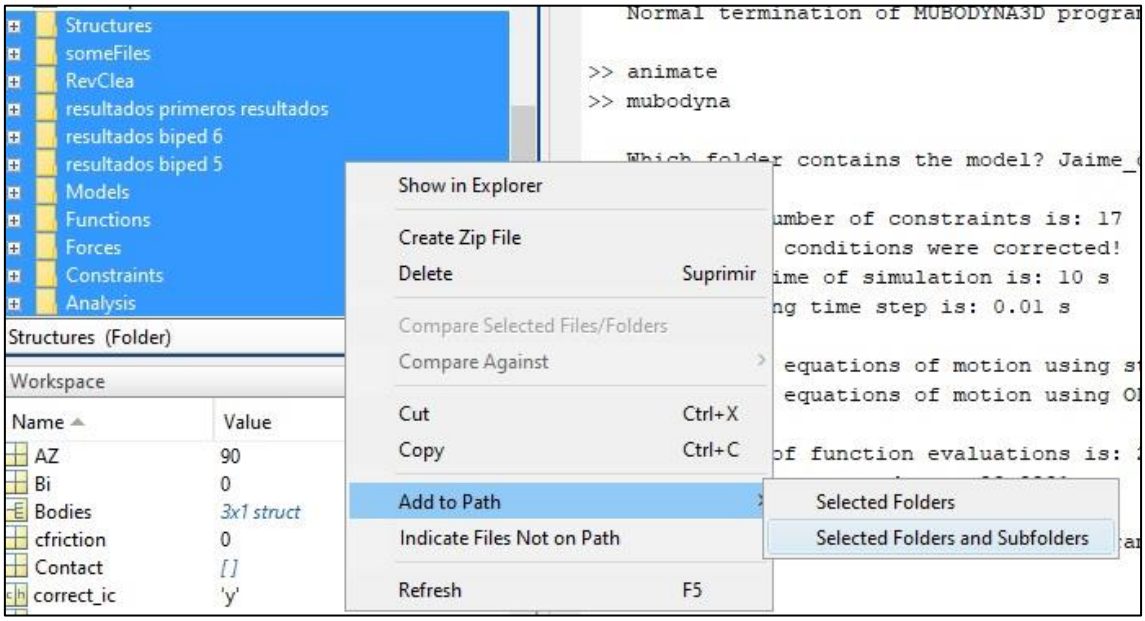

Figura 54. Add to path

En la *Figura 54* se muestran los pasos para añadir las carpetas a la ruta. Se seleccionan, se pulsa el botón derecho del ratón, y se selecciona la opción de Add to path.

```
>> animate
>> mubodyna

   Which folder contains the model? Jaime_deslizadera_final

   The total number of constraints is: 17
   The initial conditions were corrected!
   The final time of simulation is: 10 s
   The reporting time step is: 0.01 s

   Solving the equations of motion using standard method ...
   Solving the equations of motion using ODE Integrator ...

   The number of function evaluations is: 259
   The CPU time consumed was: 29.8281 s

   Normal termination of MUBODYNA3D program!

>> animate
>> save Jaime_deslizadera_final.mat
>> load Jaime_deslizadera_final.mat
>> animate
```

Figura 55. Comandos save and load

En la *Figura 55* se muestran los pasos a seguir para guardar la simulación. Se ha utilizado el modelo de la deslizadera.

CONCLUSIONES

Durante la primera parte del proyecto se ha entendido el proceso de definición de un sistema mecánico de manera general, adquiriendo los conocimientos necesarios para la definición de su sistema mecánico, y explicando conceptos y definiciones que puedan complementar el trabajo "*Dinámica por ordenador con Matlab*" [1], y sirvan de orientación en investigaciones futuras.

Durante la segunda parte se han adquirido los conocimientos de MUBODYNA, y los procesos del programa para definir y simular el comportamiento de un sistema mecánico. Se ha comprobado la enorme utilidad y precisión del programa, y su gran ámbito de aplicación, obteniendo correctas simulaciones de los sistemas modelados.

Por lo tanto, se puede concluir que se han cumplido los objetivos propuestos al principio de este, entendiendo todos los conceptos necesarios, y modelando sistemas mecánicos con el programa, de manera que puedan ser analizados y mejorados por futuros investigadores.

PRESUPUESTO DEL PROYECTO

A continuación, se va a presentar el presupuesto del proyecto:

Personal	Categoría	Tiempo (h)	Cotización (€/hora)	Coste total (€)
Eduardo Corral Abad	Doctor de ingeniería	100	30	3.000
Jaime San Martín Puy	Ingeniero Junior	400	10	4.000
TOTAL				**7.000**

Tabla 1. Personal involucrado

Hardware	Unidades	Coste unitario (€)
Ordenador portátil	1	600
Ratón	1	30
TOTAL		**630**

Tabla 2. Hardware

Software	*Unidades*	*Coste (€)*
Matlab (Licencia académica)	1	0
TOTAL		**0**

Tabla 3. Software

Por lo tanto, el presupuesto total del proyecto es de 7.630 €.

BIBLIOGRAFÍA

[1] Sergio Villar Alegría, "Dinámica por ordenador con Matlab" Trabajo de fin de grado, Departamento de Ingeniería Mecánica, Universidad Carlos III de Madrid, Madrid, España, 2018.

[2] SHABANA, A.A, "Computational Dynamics". Wiley, 2001.

[3] SHABANA, A.A "Dynamics of Multibody Systems" Ed. 4, Cambridge University, 2013.

[4] Parviz E. Nikravesh, "Computer-Aided Analysis of Mechanism Systems", Prentice Hall, 1998.

[5] Nikravesh PE "Initial condition correction in multibody dynamics", CRC, press, 2007 London.

[6] Paulo Flores, "Concept and Formulation for Spatial Multibody Dynamics", Springer 2015.

[7] Javier García de Jalón, Eduardo Bayo, "Kinematic and Dynamic Simulation of Multibody Systems", The Real Time Challenge, Springer 2009